CSCE 240: Advanced Programming Techniques
Lecture 23: Templates

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

5TH APRIL 2022

Carolinian Creed: "I will practice personal and academic integrity."

Credits: Some material reused with permission of Dr. Jeremy Lewis. Others used as cited with thanks.

Organization of Lecture 23

- Introduction Section
  - Recap of Lecture 22
  - News / announcements / clarifications
  - TA and SI Updates

- Main Section
  - Concept: Templates
  - Concept: Class Templates
  - Concept: Function Templates
  - Task: HW 6 (Optional)
  - Task: Project – PA #5 due

- Concluding Section
  - About next lecture – Lecture 24
  - Ask me anything

Introduction Section

Recap of Lecture 22

- We discussed code optimization considerations
  - Memory optimization
  - Runtime optimization
  - Code maintenance ease

- Looked at examples
  - Sorting
  - Searching
  - Project

Announcement

- McNair Junior Fellows program: **30 grantees** this summer, and we sure hope you can encourage your students to explore this opportunity. All details and applications are on: http://www.cec.sc.edu/mjf | **Deadline April 8th, 2022!**
  - The program, in its 8th year since its foundation, and in its 4th year as an official CEC program, provides supports for undergraduate students up to 3k$ in summer funds and runs activities that helps the students further explore research (as well as research posters, state of the art and other research initiation programs).
  - Contact: Ramy Harik

- Summer Internships at AI Institute
  - You can work with faculty and get paid
  - You can apply to fellowship as well as work with faculty (with/ without pay)
  - You can work on your idea with a faculty to mentor (with/ without fellowship)

Updates from TA, SU

- TA update: Yuxiang Sun (Cherry)

- SI update: Blake Seekings

Guidance on Readme

Code Structure for any Assignment

- File: Readme.md   // (or Readme.txt or readme.md)
                    // Has information about – who, what, why, how
                    // - In our case: code for course and PA#, author, layout of sub-folders, etc
                    // - (optional) have readme in major sub-dirs as needed

- ./data   // Data needed or created by the program
                    // - (optional) have sub-folder for input and output, if both present
                    // - (optional) others as logically needed

- ./doc   // Document assumptions, algorithm

- ./src   // (or ./code) – Has the program source code
                    // - (optional) divide further into header files, source code,

- ./test   // Has information about how to test your code
                    // - a transcript of code's running and its input/ output
                    // - (optional) a code to check the working of code on a test case
                    // - (optional) test cases

Feedback on PA4 (from TA)

Feedback: Folders Management

1. Write a good readme

2. Make your project well-organized

3. Configure the class path well

Feedback: Feature Implementation Methods

Input reprocessing
Step 1: splitting the input into words by identifying space
Step 2: converting all letters into upper case or lower case

Method 2: regular expression

```
//block of regex patterns to search each line for
Pattern namePattern = Pattern.compile("(?<=Representative )((\\w|\\.)+ ?){0,3}");
Pattern partyPattern = Pattern.compile("Republican|Democrat");
Pattern districtPattern = Pattern.compile("(District )\\d+");
```

Method 1: looking up keyword and computing confidence
Coding implementation:
Two loops:
- outer is for determining which question is being asked
- inner: if one question is asked appropriately
Pros: straightforward
Cons: lack of flexibility, slow

Method 3: advanced comparison algorithm
-Levenshetein distance (character-level): it can compare two words distance
e.g., "rep" vs "representative", "information" vs "info"
- Prefix tree (tire): for big searching engine, e.g. Google

Takeaway: make sure your program can answer simple questions first, and then try with more sophisticated algorithms.

Feedback: More on Features

Question rephrasing:
-- Requiring the rephrased question can pass confidence;
-- interaction with user by giving hint:
"I assume you are asking for the contact information, Y/N?"

Confidence computing:
-- counting by one point if one word matched;
-- different weights
 e.g. allocating more weight to "committee" in the sixth question

Codes reusability:
-   Put individual classes into separate head files;
-   Think about the reusability issue before coding;

Data Structure: for storing read answers
-   2D array
-   Nested dictionary   (Jaya, Python)

Language filter: to deal with non-English character

Other issues found in PA4:
-The users' questions are repeated multiple times;
- Can't quit successfully

Main Section

Goals of Templates

- Generalize on coding best practices

- Improved developer productivity

- Without impacting code runtime

Steps for Using Templates

- The programmer creates functions with templates

- The compiler (effectively) creates specific functions for different types the functions are invoked with

- The user calls the functions (mostly) seamlessly

Concept: Function Templates

Simple Template

- Code example

```
template <class T>
bool less_than(T a, T b) {
        return a < b;
}
```

```
template <typename T>
bool less_than(const T& a, const T& b) {
        return a < b;
}
```

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 19

Medium (-ly Complex) Template

- Code example
  - Option 0

```cpp
template <typename T>
T sum(const vector<T>& v) {
        T result = 0;
        for (T elem : v)
                result = result + elem;
        return result;
}
```

Credit: Fundamentals of Programming C++, Richard L. Halterman, Chap. 19

Concept: Class Templates

Class Template

- Generic classes which do encapsulation of similar capability
  - Data members
  - Functions

- Commonly used to implement new data structures

- Code example
  - Options 2 and 3

Code Example

- Code example
  - Options 2 and 3

```cpp
// Class template with single parameter
template <class A, class B>
class SingleClassTemplate {
    private:
     A a;

    public:
     SingleClassTemplate(A aa) : a(aa)
        {}  // constructor


     void printValues() {
         cout << "\ta = " << a << endl;
     }
};
```

Credit: Adapted from
https://www.programiz.com/cpp-programming/class-templates

Discussion

- Templates are meant to increase developer productivity

- One can define for functions or classes

- Can have one or more types

- Compiler generates type-specific code; hence, little-to-no impact on code performance

- Be aware of the initial values and operators being defined for the types
  - Unexpected errors may happen

Class Exercise – 10 Mins

- **Objective**: Discuss where we can use in our homeworks and assignments

- Function templates

- Class templates

Home Work 6   (Optional)

Due Thursday, April 7, 2022

Home Work (#6) – C++ - Understand Code Optimization

- Consider BubbleSort algorithm for sorting (shown on right)

- Processing
  1. Generate n random numbers – called S
  2. (Store and Sort as Array)
     1. Store the numbers of S in an array: allocated with size n at the start
     2. Sort using Bubble Sort
  3. (Store and Sort as Vector)
     1. Store the numbers of S in a vector: allocated with size 1 and size increased one by one number until n
     2. Sort using Bubble Sort
  4. Measure time difference in both cases with n = 100, 1000, 10000, 100000

- Output
  1. Make a graph showing any difference in timing
  2. Check the sorted results and confirm they are same

Algo 3 (BubbleSort):

- current_array = a = Input
- For (i=0; i<**N-1**; i++) {
  - For (**j=(i+1)**; j<N; j++) {
    - If(a[i] > a[j])
      - Swap(a[i], a[j])
  - }
- }
- **Return current_array**

Home Work (#6) – C++ - Requirement

- So, write a program named:
  SortRandomN

- It will support inputs/ arguments in two formats:
  - n: number // numbers to sort

- Output:
  - Time for array   // computed value
  - Time for vector // computed value
  - Result_checked (Boolean) = True/ False
    // compare the results in both cases

Example invocation

> **SortRandomN** 1000
Time for array: 1 sec
Time for vector: 2 sec
Result_checked: True

Discussion: Course Project

Course Project – Assembling of Prog. Assignments

- **Project**: Develop collaborative assistants (chatbots) that offer innovative and ethical solutions to real-world problems ! *(Based on competition - https://sites.google.com/view/casy-2-0-track1/contest)*

- Specifically, **the project will be building a chatbot that can answer questions about a South Carolina member of state legislature from**:
https://www.scstatehouse.gov/member.php?chamber=H

  - Each student will choose a district (from 122 available).
  - Programming assignment programs will: (1) extract data from the district, (2) process it, (3) make content available in a command-line interface, (4) handle any user query and (5) report on interaction statistics.

Core Programs Needed for Project

- Prog 1: extract data from the district **[prog1-extractor]**

- Prog 2: process it (extracted data) based on questions **[prog2processor]**

- Prog 3: make content available in a command-line interface **[prog3-ui]**

- Prog 4: handle any user query **[prog4-userintent2querymapper]**

- Prog 5: report statistics on interaction of a session, across sessions **[prog5-sessionlogger]**

Objective in Programming Assignment # 5:
Record what happens in a chat session and provide summary

• A user may interact with your chatbot for one question or twenty. How did your chatbot do?

• **Record chat your system makes with each user and report on user session as well total usage statistics (since the chatbot was created)**

Approach Suggested
• Under data folder,
  • have a sub-folder called **chat_sessions**
    • When a person starts a chat session (i.e., starts your program and until does not quit), create a file with the " <data>_<time>.txt" as the name. Save the user's utterance and the system's reply there in the order they come. Close this file when the user session ends.
    • Calculate statistics: # user_utterance, #system_utterance and time duration of session
  • have a file called **chat_statistics.csv**.
    • Have a header with columns: S.No, chat_file, # user_utterance, #system_utterance and time taken
    • For each chat file in chat_sessions, there will be a row with the chat statistics you have calculated

Objective in Programming Assignment # 5:
Record what happens in a chat session and provide summary

Approach Suggested
- Under data folder,
  - have a sub-folder called **chat_sessions**
    - When a person starts a chat session (i.e., starts your program and until does not quit), create a file with the " <data>_<time>.txt" as the name. Save the user's utterance and the system's reply there in the order they come. Close this file when the user session ends.
    - Calculate statistics: # user_utterance, #system_utterance and time duration of session
  - have a file called **chat_statistics.csv**.
    - Have a header with columns: S.No, chat_file, # user_utterance, #system_utterance and time taken
    - For each chat file in chat_sessions, there will be a row with the chat statistics you have calculated

- Goal: report statistics on interaction of a session, across sessions [Name: **prog5-sessionlogger**]

- One can invoke it with arguments
  - **prog5-sessionlogger –summary**
    - There are 12 chats to date with user asking 23 times and system respond 24 times. Total duration is 456 seconds.
  - **prog5-sessionlogger –showchat-summary 2**
    - Chat 2 has user asking 2 times and system respond 2 times. Total duration is 4 seconds.
  - **prog5-sessionlogger –showchat 2**
    - Chat 2 chat is:

      ...
  - **prog5-sessionlogger –showchat 200**
    - ERROR: there are only 12 chat sessions. Please choose a valid number.

Programming Assignment # 5

- Code organization
  - Create a folder in your GitHub called "**prog5-sessionlogger**"
  - Have sub-folders: src (or code), data, doc, test
  - Have data directory as shown in previous slide
    - ./data/**chat_sessions/**
    - ./data/ **chat_statistics.csv**
  - Write a 1-page report in ./doc sub-folder
  - Put a log of system interacting in ./test
  - Send a confirmation that code is done by updating Google sheet; optionally, send email to instructor and TA

- Use concepts learned in class
  - Exceptions
  - File operations
  - Dynamic memory

Concluding Section

Lecture 23: Concluding Comments

- Programming practice for project assignments based on PA#4

- We discussed
  - Templates
  - Class templates
  - Functional templates

- HW 6 (optional) – due on Thursday

About Next Lecture – Lecture 24

Lecture 24: AI/ ML

- AI as a decision-support

- ML, and the DL craze

- AI/ML and programming – what to be aware of

20	Mar 24 (Th)	Advanced: Operator overloading	Prog 4 - end
21	Mar 29 (Tu)	Advanced: Memory Management	Prog 5 - start
22	Mar 31 (Th)	Advanced: Code efficiency	
23	Apr 5 (Tu)	Advanced: Templates	Prog 5 – end, Optional HW-6
24	Apr 7 (Th)	AI / ML and Programming	Prog 6 - assembling
25	Apr 12 (Tu)	Review material for Quiz 2	Project due
26	Apr 14 (Th)	In class test	Quiz 2 – In class